Django Admin

Apresentação

O Django é um *framework* de alto nível, escrito em Python, que encoraja o desenvolvimento limpo de aplicações Web, e tem sido empregado em milhares de aplicações, desde aplicações robustas até aplicações de menor complexidade.

Nesta Unidade de Aprendizagem, você conhecerá o Django Admin, identificando suas vantagens e funcionalidades, além da organização
do seu painel administrativo.

Bons estudos.

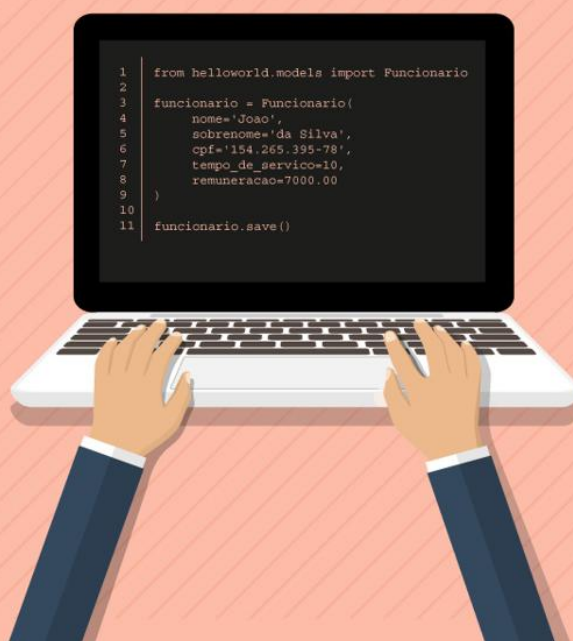
Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Justificar o Django Admin.
- Identificar as vantagens e as funcionalidades do Django Admin.
- Organizar o painel administrativo.

Uma das grandes vantagens da utilização do Django é a facilidade de abstração na implementação e instalação de comandos, podendo, inclusive, utilizar o próprio prompt de comando para instalações diversas. Sendo assim, imagine a seguinte situação:

Você é desenvolvedor e trabalha para uma multinacional do ramo de integração de dados em Python, por meio do Django.

Recebeu como tarefas certificar-se de que o Python e o PIP (gerenciador de pacotes do Python) estão instalados corretamente, e realizar a instalação do Django por meio do *prompt* de comando.

```
1  from helloworld.models import Funcionario
2
3  funcionario = Funcionario(
4        nome='Joao',
5        sobrenome='da Silva',
6        cpf='154.265.395-78',
7        tempo_de_servico=10,
8        remuneracao=7000.00
9  )
10
11 funcionario.save()
```

Explique como você se certificaria da instalação do Python e do PIP, além de como faria a instalação do Django via prompt de comando.

O Django é um framework muito escalável: foi desenvolvido para tirar vantagem da maior quantidade de *hardware* possível. Pensando nisso, a arquitetura do Django foi segmentada em três camadas interdependentes: *Model*, *View e Template.*

Neste Infográfico, você vai ver, esquematicamente, o fluxo da arquitetura de integração de cada uma dessas camadas, assim como suas requisições e respostas por meio do protocolo HTTP.

ARQUITETURA DE INTEGRAÇÃO DJANGO PYTHON

O Django utiliza a linguagem Python no arcabouço da sua formação e a arquitetura MVT (*Model-View-Template*):

- **Camada *Model*:** responsável pelo modelo de negócio da aplicação. Realiza as regras de negócio da aplicação.
- **Camada *View*:** responsável pelo controle das requisições. Realiza o controle das requisições feitas e orquestra as telas de visualização com o modelo de negócio.
- **Camada *Template*:** responsável pela interface com o usuário. Realiza requisições e recebe respostas via HTTP.

O diagrama a seguir apresenta, de forma esquematizada, essas camadas de integração:

CAMADA *TEMPLATE*

Website

Requisição via HTTP

CAMADA *VIEW*

Resposta via HTTP

Roteamento de URLs

urls.py

Views e *Middlewares*

views.py

HTML CSS JS

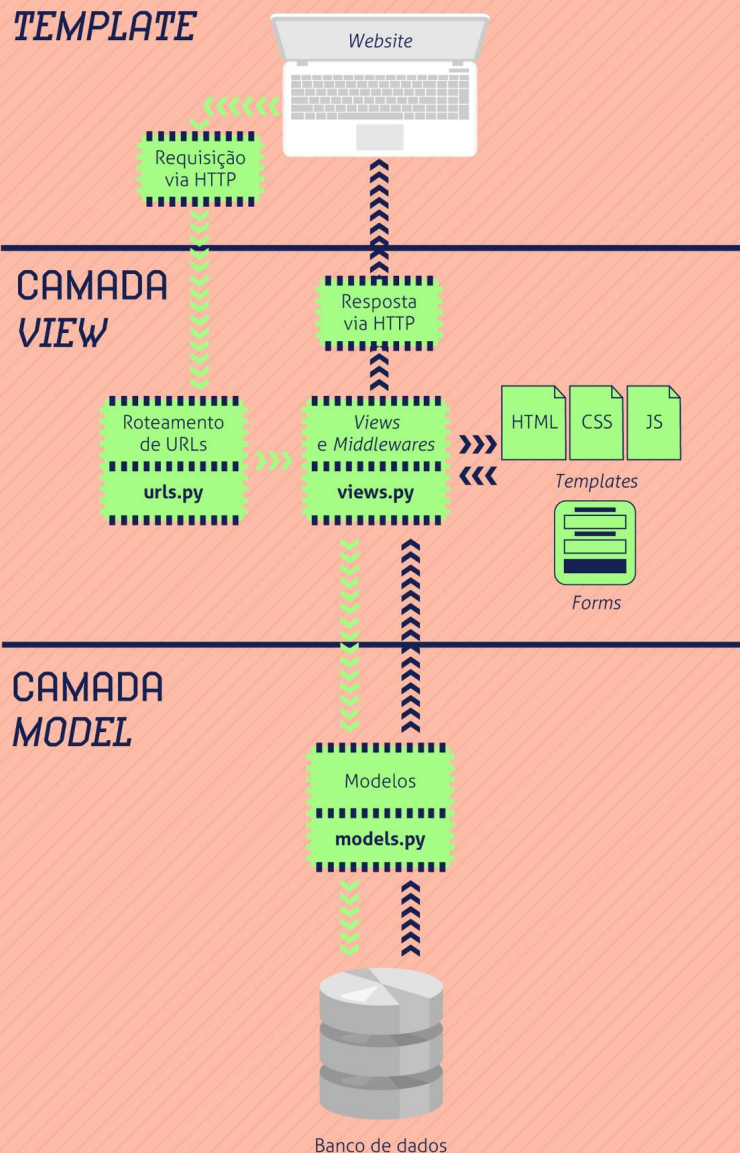
Templates

Forms

CAMADA *MODEL*

Modelos

models.py

Banco de dados

Fluxo de requisição

Fluxo de resposta

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

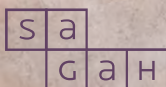
Conteúdo do livro

O Django é um *framework* responsável pela parte do desenvolvimento Web, como tratamento de requisições, mapeamento objeto-relacional e preparação de respostas HTTP, para que, dessa forma, sejam gastos esforços com aquilo que realmente importa nas aplicações Web, isto é, as regras de negócio das aplicações.

No capítulo Django Admin, da obra *Programação Back End I*, base teórica desta Unidade de Aprendizagem, você vai aprender sobre as principais características do Django, suas funcionalidades arquiteturais e vantagens. Por fim, vai ver a organização do seu painel administrativo.

Boa leitura.

PROGRAMAÇÃO BACK END I

Pedro Henrique Chagas Freitas

sagah SOLUÇÕES EDUCACIONAIS INTEGRADAS

Django Admin

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Justificar o Django Admin.
- Identificar as vantagens e funcionalidades do Django Admin.
- Organizar o painel administrativo.

Introdução

O Django Admin é um *framework* Python de alto nível, que incentiva o desenvolvimento coeso de aplicações web, por meio da própria linguagem Python. É responsável pela parte de tratamento de requisições, mapeamento objeto-relacional, preparação de respostas HTTP, por exemplo, para que, dessa forma, esforços sejam concentrados naquilo que realmente importa nas aplicações Web, isto é, as regras de negócio.

Neste capítulo, vamos abordar os conceitos principais do Django Admin, identificar suas vantagens e funcionalidades e organizar seu painel administrativo.

Introdução do Django Admin

O Django Admin é um *framework* bastante escalável, desenvolvido para tirar vantagem da maior quantidade de *hardware* possível. Django usa uma arquitetura "zero-compartilhamento", o que significa que, na prática, podemos adicionar mais recursos em qualquer nível — servidores de banco de dados, cache e/ou servidores de aplicação (TONSIG, 2008).

Ele foi desenvolvido com uma preocupação extra em segurança, evitando os mais comuns ataques — como *cross site scripting* (XSS), *cross site request forgery* (CSRF), *SQL injection*, entre outros —, com o objetivo de construção de aplicações web em Python. De acordo com sua documentação, os desenvolvedores o declaram como um *framework* MTV, isto é, *model-template-view*.

Para os desenvolvedores, as *views* do Django representam a informação que você vê, não como você vê. Há uma sutil diferença: no Django, uma *view* é uma forma de processar os dados de uma URL específica, pois ela descreve a informação que é apresentada por meio do processamento descrito pelo desenvolvedor em seu código. Além disso, é imprescindível separar conteúdo de apresentação no Django e em qualquer outro *framework* — que é onde os *templates* residem.

Como dito, uma *view* descreve a informação apresentada, mas normalmente delega para um *template*, que descreve como a informação é apresentada. Assim, onde temos o *controller* nessa arquitetura? No caso do Django, é o próprio *framework* que faz o trabalho pesado de processar e rotear uma requisição para a *view*, conforme a configuração de URL descrita pelo desenvolvedor (PRESSMAN, 2011).

Para ajudar a entender um pouco melhor, vamos analisar ao longo deste capítulo a saída de uma requisição do *browser* do usuário, passando para o servidor onde o Django está sendo executado e retornando ao *browser* do usuário. Faremos isso em conjunto com a apresentação de suas funcionalidades e vantagens.

Vantagens e funcionalidades

A maior vantagem do Django está diretamente ligada às suas facilidades, isto é, sua conceituação arquitetural dividida em três camadas: de *models*, *views* e de *templates*.

A camada de modelos tem uma função essencial na arquitetura das aplicações desenvolvidas com o Django. É nela que descrevemos os campos e comportamentos das entidades que comporão nosso sistema, e onde reside a lógica de acesso aos dados da nossa aplicação (SOMMERVILLE, 2008).

Indo um pouco mais fundo, temos a camada *model* da arquitetura MTV do Django (*model-template- view*). Nela, vamos descrever, em forma de classes, as entidades do nosso sistema, para que o resto (*template* e *view*) possa interoperar.

Um modelo é a descrição do dado que será gerenciado pela sua aplicação Django. Ele contém os campos e comportamentos desses dados. No fim, cada modelo equivalerá a uma tabela no banco de dados. No Django, um modelo tem basicamente duas características:

- é uma classe que herda de `django.db.models.Model`;
- cada atributo representa um campo da tabela.

Com isso, Django gera automaticamente uma API de acesso a dados, a qual facilita, e muito, a administração do painel do Django, no momento de gerenciamento (adicionar, excluir e atualizar) dos dados. Para entendermos melhor, vamos modelar nosso "Hello, World"!

Vamos supor que sua empresa está desenvolvendo um sistema de gerenciamento dos funcionários, e lhe foi dada a tarefa de modelar e desenvolver o acesso aos dados da entidade funcionário. Pensando calmamente em sua estação de trabalho, enquanto seu chefe lhe cobra diversas metas e diz que o *deadline* do projeto foi adiantado em duas semanas, você pensa nos seguintes atributos para tal classe (TONSIG, 2008).

- nome;
- sobrenome;
- CPF;
- tempo de serviço;
- remuneração.

Agora, é necessário passar isso para código Python, a fim de que o Django possa entender (lembre-se de que Django é implementado na linguagem Python). No Django, os modelos são descritos no arquivo `models.py`. Supomos que ele já foi criado e está presente na pasta `helloworld/models.py`.

Temos, então, que descrever somente cada atributo (nome, sobrenome, CPF, etc.) como um campo (ou *field*) da nossa classe de modelo, a qual chamaremos de funcionário. Seguindo as duas características que apresentamos (herdar da classe *model* e mapear os atributos da entidade com os campos), podemos descrever nosso modelo da seguinte forma:

```
1   from django.db import models
2
3   class Funcionario(models.Model):
4
5     nome = models.CharField(
6       max_length=255,
7       null=False,
8       blank=False
9     )
10
11    sobrenome = models.CharField(
12      max_length=255,
13      null=False,
14      blank=False
15    )
16
17    cpf = models.CharField(
18      max_length=14,
19      null=False,
20      blank=False
21    )
22
23    tempo_de_servico = models.IntegerField(
24      default=0,
25      null=False,
26      blank=False
27    )
28
29    remuneracao = models.DecimalField(
30      max_digits=8,
31      decimal_places=2,
32      null=False,
33      blank=False
34    )
35
36    objetos = models.Manager()
```

Agora, vejamos a camada *view*, na qual descreveremos a lógica de negócios da nossa aplicação, ou seja, essa camada tem a responsabilidade de processar as requisições vindas dos usuários, formar uma resposta e enviá-la de volta a ele. Temos, na *view*, a nossa lógica de negócio. A camada *view* deve: recepcionar, processar e responder. Para isso, começamos pelo roteamento de URLs.

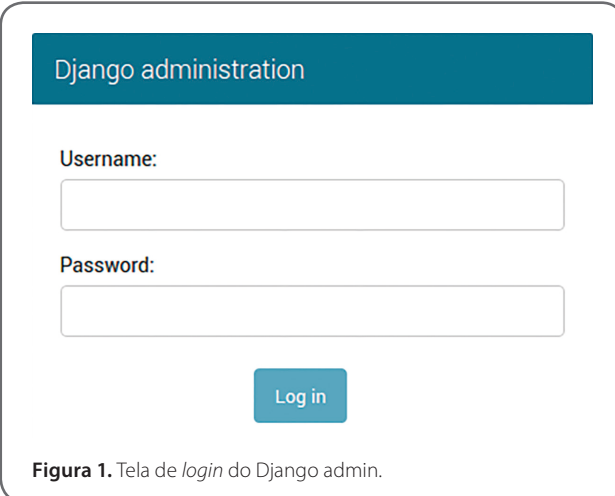
A partir da URL que o usuário quer acessar (`/funcionarios`, por exemplo), o Django roteará a requisição para quem for tratá-la. Mas, primeiramente, o Django precisa ser informado para onde mandar a requisição. Fazemos isso no arquivo chamado URLConf e damos o nome, por convenção, de `urls.py`.

Portanto, devemos criar um arquivo `urls.py` dentro da pasta `/hello-world` e outro na pasta `/website`. Como o app *helloworld* é o núcleo da nossa aplicação, ele faz o papel de centralizador de rotas, isto é, primeiro, a requisição cai no arquivo `/helloworld/urls.py` e é roteada para o app correspondente; em seguida, o URLConf do app (`/website/urls.py`, no nosso caso) roteará a requisição para a *view* que processará a requisição. Dessa forma, o arquivo `helloworld/urls.py` deve conter:

```
1  from django.urls.conf import include
2  from django.contrib import admin
3  from django.urls import path
4
5  urlpatterns = [
6    # Inclui as URLs do app 'website'
7    path('', include('website.urls', namespace='website')),
8
9    # Interface administrativa
10   path('admin/', admin.site.urls),
11 ]
```

Assim, o Django tentará fazer o *match* de URLs, primeiro, no arquivo de URLs do *app* `website` (`website/urls.py`) e, depois, no URLConf da plataforma administrativa. Agora, vamos falar da camada *template*.

A camada *template* é que dá cara à aplicação, isto é, faz a interface com o usuário. É nela que se encontra o código Python, responsável por renderizar nossas páginas Web, e os arquivos HTML, CSS e JavaScript, que darão vida à aplicação. Essa é a camada que faz a interface do nosso código Python/Django com o usuário, interagindo, trocando informações, captando dados de *input* e gerando dados de *output* (PRESSMAN, 2011).

Lembre-se de que um *template* é um arquivo de texto que pode ser transformado em outro arquivo (um arquivo HTML, um CSS, um CSV, etc.). Um *template* no Django contém variáveis que podem ser substituídas por valores, a partir do processamento por uma *engine* de *templates* (núcleo ou "motor" de *templates*) — usamos o marcador `{{ variável }}`. *Tags* controlam a lógica do *template* — `{% tag %}`. Filtros adicionam funcionalidades ao *template* — `{{ variável|filtro }}`.

Por exemplo, a seguir, está representado um *template* mínimo que demonstra alguns conceitos básicos:

```
1  {# base.html contém o template que usaremos como esqueleto #}
2  {% extends "base.html" %}
3
4  {% block conteudo %}
5   <h1>{{ section.title }}</h1>
6
7   {% for f in funcionarios %}
8    <h2>
9     <a href="{% url 'website:funcionario_detalhe' pk=f.id %}">
10     {{ funcionario.nome|upper }}
11    </a>
12   </h2>
13  {% endfor %}
14 {% endblock %}
```

Para facilitar a manipulação de *templates*, os desenvolvedores do Django criaram uma linguagem que contém todos esses elementos. Chamaram-na de DTL — *Django template language*.

Painel administrativo

Para acessar o painel administrativo do Django Admin, precisamos, primeiramente, criar um usuário e uma senha, conforme Figura 1, a seguir.

Figura 1. Tela de *login* do Django admin.

Para criar o usuário admin, vamos executar o seguinte comando e informar o nome de usuário, *e-mail* e senha que escolhemos.

```
$ python manage.py createsuperuser
Username (leave blank to use 'currentuser'): admin
Email address: aluno@gmail.com
Password: 123456
Password (again): 123456
Superuser created successfully.
```

Em seguida, é gerada uma tela semelhante à da Figura 2, a seguir, que é o painel administrativo do Django.

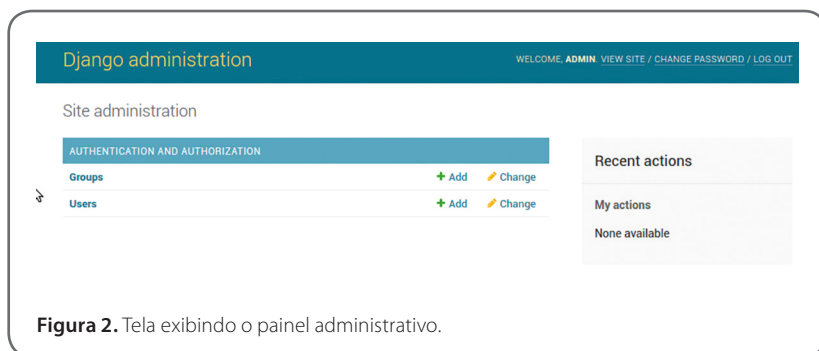



Figura 2. Tela exibindo o painel administrativo.

Você acessará, também, as telas administrativas de usuários e grupos de usuário, como na Figura 3, a seguir.

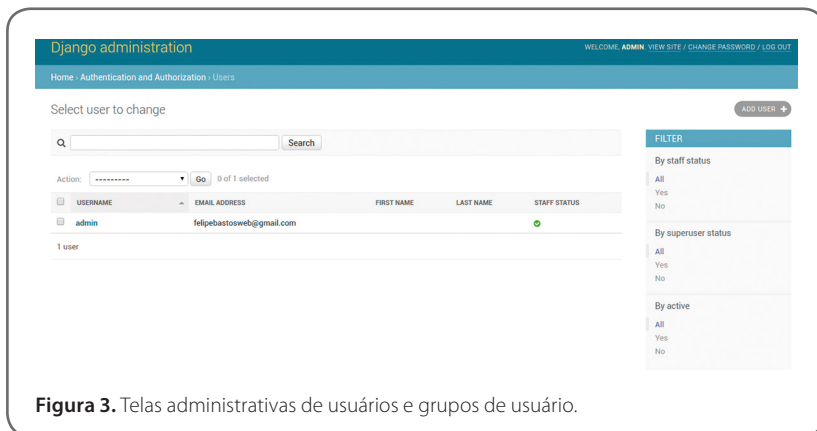



Figura 3. Telas administrativas de usuários e grupos de usuário.

Dentre as finalidades dessa tela, temos a administração dos dados na arquitetura MTV, seu fluxo de vida e, principalmente, a criação de novos usuários comuns e grupos de usuários.

Este capítulo mostrou o grande valor agregado e a importância do *framework* Django para a linguagem Python e para o desenvolvimento coeso de aplicações Web.

Referências

PRESSMAN, R. S. *Engenharia de software*: uma abordagem profissional. 7. ed. Porto Alegre: AMGH; Bookman, 2011. 780 p.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Pearson Prentice Hall, 2008. 552 p.

TONSIG, S. L. *Engenharia de software*: análise e projeto de sistemas. 2. ed. Rio de Janeiro: Ciência Moderna, 2008. 319 p.

Dica do professor

O Django faz as operações pesadas do desenvolvimento Web. Nesse contexto, é possível utilizar funções no tratamento de requisições, o qual contempla a preparação de respostas HTTP.

Nesta Dica do Professor, você vai ver sobre o uso de funções para criação de requisições HTTP com POST e GET via Django.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

1) **O Django é um *framework* Python de alto nível, que incentiva o desenvolvimento coeso de aplicações Web por meio da própria linguagem Python. É responsável pela parte do desenvolvimento**
Web, como tratamento de requisições, mapeamento objeto-relacional, entre outros.

Sendo assim, é correto afirmar que o Django é um *framework*
de arquitetura:

A) MVC (*Model-View-Controller*).

B) MCT (*Model-Controller-Transfer*).

C) MTV (*Model-Template-View*).

D) MTA (*Model-Transfer-Abstract*).

E) MVT (*Model-View-Trought*).

2) **No Django, uma *View* é uma forma de processar os dados de uma URL específica, pois ela descreve qual informação é apresentada, por meio do processamento descrito pelo desenvolvedor em seu código.**

Além disso, é imprescindível separar conteúdo de:

A) apresentação.

B) negócio.

C) abstração.

D) gestão.

E) orquestração.

3) **A maior vantagem do Django está diretamente ligada às suas facilidades, isto é, à sua conceituação arquitetural em três camadas.**

A camada de Modelos tem uma função essencial na arquitetura das aplicações desenvolvidas com o Django, pois é nela que se descreve(m):

A) os campos e os comportamentos das entidades que não irão compor o sistema, e a lógica de acesso aos dados da aplicação.

B) os comportamentos das entidades que irão compor o sistema, unicamente.

C) a lógica de acesso aos dados da aplicação, unicamente.

D) a lógica de acesso aos dados da aplicação e os campos e comportamentos das entidades que irão compor o sistema.

E) o *template* da lógica de acesso e os comportamentos das entidades externas ao sistema.

4) **Um modelo é a descrição do dado que será gerenciado pela sua aplicação Django e contém os campos e comportamentos desses dados. No fim, cada modelo vai equivaler a uma tabela no banco de dados.**

No Django, um modelo tem basicamente duas características, que são:

A) o fato de cada atributo representar um campo da tabela e de ser uma classe que herda de django.db.models.View.

B) o fato de cada atributo representar um campo da tabela e de ser uma classe que herda de django.db.models.Template.

C) o fato de cada atributo representar um campo da tabela e de ser uma classe que herda de django.db.views.Model.

D) o fato de cada atributo representar um campo da tabela e de ser uma classe que herda de django.db.template.Model.

E) o fato de cada atributo representar um campo da tabela e de ser uma classe que herda de django.db.models.Model.

5) **Entre as finalidades do painel de administração do Django, tem-se a administração dos dados na arquitetura MTV, seu fluxo de vida e, principalmente:**

A) a criação de novos usuários comuns e grupos de usuários.

B) a criação de novas classes e métodos.

C) a criação de novos atores e relacionamentos.

D) a criação de novas instâncias.

E) a criação de novos parâmetros.

Na prática

Uma das funções mais utilizadas pelo framework Django é a criação de apps, que são estruturas modulares de projetos usadas para organizar e separar funções específicas da aplicação. Um exemplo muito comum é a criação de API de acesso a dados em empresas.Neste Na Prática, você vai ver como é simples manipular os dados de um sistema por meio da poderosa API de acesso a dados do Django.

API DE ACESSO A DADOS DO DJANGO

Marcos está desenvolvendo um sistema de gerenciamento de funcionários e precisa modelar e desenvolver o acesso aos dados da entidade Funcionário.
A partir da classe Funcionário modelada, para facilitar o processo, ele utilizou a API de acesso a dados provida pelo Django.

Assim, testou a adição de um novo funcionário utilizando o *shell* do Django. Para isso, digitou o comando:

```
1  python manage.py shell
```

Para adicionar um novo funcionário, criou uma instância do seu modelo e chamou o método **save()**. Para isso, utilizou o seguinte código (no *shell* do Django):

```
1   from helloworld.models import Funcionario
2
3   funcionario = Funcionario(
4        nome='Joao',
5        sobrenome='da Silva',
6        cpf='154.265.395-78',
7        tempo_de_servico=10,
8        remuneracao=7000.00
9   )
10
11  funcionario.save()
```

Assim, o funcionário João da Silva foi salvo no seu banco de dados.

VANTAGENS

Nada de código SQL e *queries* enormes.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Ambiente de desenvolvimento: aprenda Django

Para compreender melhor o funcionamento da estrutura de implementação do Django em ambiente Python, veja o vídeo a seguir.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Django 2.0 em 30 minutos

O Django 2.0 trouxe muitas novidades, e a principal delas é o suporte exclusivo de Python 3.4. No vídeo a seguir, você vai acompanhar uma contextualização sobre o Django 2.0 e suas funcionalidades de administração.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Para que serve o Django?

Você sabe o que faz o Django? Neste breve vídeo, você vai conhecer as funcionalidades e a abrangência do Django com linguagem Python.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Criando um projeto, tabelas e utilizando o Django Admin

No vídeo a seguir, você vai acompanhar a construção de uma plataforma de adoção de *pets*, na qual pessoas podem consultar e cadastrá-los para adoção.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.